



XMPP

XEP-0392: Consistent Color Generation

Jonas Schäfer

<mailto:jonas@wielicki.name>

<xmpp:jonas@wielicki.name>

2019-10-16

Version 0.7.0

Status	Type	Short Name
Experimental	Standards Track	colors

This specification provides a set of algorithms to consistently generate colors given a string. The string can be a nickname, a JID or any other piece of information. All entities adhering to this specification generate the same color for the same string, which provides a consistent user experience across platforms.

Legal

Copyright

This XMPP Extension Protocol is copyright © 1999 – 2020 by the [XMPP Standards Foundation](#) (XSF).

Permissions

Permission is hereby granted, free of charge, to any person obtaining a copy of this specification (the "Specification"), to make use of the Specification without restriction, including without limitation the rights to implement the Specification in a software program, deploy the Specification in a network service, and copy, modify, merge, publish, translate, distribute, sublicense, or sell copies of the Specification, and to permit persons to whom the Specification is furnished to do so, subject to the condition that the foregoing copyright notice and this permission notice shall be included in all copies or substantial portions of the Specification. Unless separate permission is granted, modified works that are redistributed shall not contain misleading information regarding the authors, title, number, or publisher of the Specification, and shall not claim endorsement of the modified works by the authors, any organization or project to which the authors belong, or the XMPP Standards Foundation.

Warranty

NOTE WELL: This Specification is provided on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE.

Liability

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall the XMPP Standards Foundation or any author of this Specification be liable for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising from, out of, or in connection with the Specification or the implementation, deployment, or other use of the Specification (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if the XMPP Standards Foundation or such author has been advised of the possibility of such damages.

Conformance

This XMPP Extension Protocol has been contributed in full conformance with the XSF's Intellectual Property Rights Policy (a copy of which can be found at <https://xmpp.org/about/xsf/ipr-policy>) or obtained by writing to XMPP Standards Foundation, P.O. Box 787, Parker, CO 80134 USA).

Contents

1	Introduction	1
2	Requirements	1
3	Use Cases	1
3.1	Generating a color	1
4	Business Rules	2
5	Algorithms	2
5.1	Angle generation	2
5.2	Corrections for Color Vision Deficiencies	3
5.2.1	Red/Green-blindness	3
5.2.2	Blue-blindness	3
5.3	RGB generation	3
5.4	Conversion of an RGB color palette to a Hue palette	3
5.5	Mapping of a Hue angle to closest palette color	4
6	Implementation Notes	4
6.1	Gamma Correction	4
6.2	Normalization	5
7	Accessibility Considerations	5
7.1	Color Vision Deficiencies	5
7.2	Contrast Ratio	5
8	Security Considerations	5
9	Design Considerations	6
9.1	The YCbCr color space	6
9.2	Hue-Saturation-Value/Lightness color space	6
9.3	Palette-based and context-aware coloring	6
9.4	Choice of mixing function in angle generation	6
9.5	Palette-mapping function	7
10	IANA Considerations	7
11	XMPP Registrar Considerations	7
12	Acknowledgements	8
13	Test Vectors	8
13.1	Test Vectors	8
13.1.1	No Color Vision Deficiency correction	8

13.1.2	With Red/Green-blindness correction	8
13.1.3	With Blue-blindness correction	8
13.2	Test Vectors for mapping to 216 color palette	9
13.2.1	No Color Vision Deficiency correction	9
13.2.2	With Red/Green-blindness correction	9
13.2.3	With Blue-blindness correction	9

1 Introduction

Colors provide a valuable visual cue to recognize shapes. Recognition of colors works much faster than recognition of text. Together with the length and overall shape of a piece of text (such as a nickname), a color provides a decent amount of entropy to distinguish a reasonable amount of entities, without having to actually read the text.

Clients have been using randomly or deterministically chosen colors for users in multi-user situations for a long time already. However, since there has been no standard for how this is implemented, the experience differs across platforms. The goal of this XEP is to provide a uniform, platform-independent, stateless and easy-to-implement way to map arbitrary bytestrings to colors.

To allow cross-client use, it is important that the color scheme can be adapted to different environments. This specification provides means to adapt colors to different background colors as well as Color Vision Deficiencies.

In no way is the system presented in this specification a replacement for names. It only serves as an additional visual aid.

2 Requirements

The color generation mechanism should provide the following features:

- Consistent generation of color across all platforms depending solely on the identifier used as input for the algorithm.
- The system should be reasonably fast; it must be possible to, for example, apply it to all roster entries even of very large rosters in reasonable amount of time.
- It must be able to provide decent contrast on any background.
- The implementation should be stateless and not be complex.
- A fallback path for users with common types of Color Vision Deficiencies must be provided.
- A fallback path for systems which can only use colors from a restricted palette must be provided.

3 Use Cases

3.1 Generating a color

To generate a color from a string of text, the following algorithms are applied in order:

1. Generate a Hue value from the text.
2. If enabled, apply configured corrections for Color Vision Deficiencies.
3. If constraints mandate the use of only a small palette of colors, map the angle to the closest palette color. (Such situations could for example be a UI environment with guidelines to only use a specific set of colors or an output device which only supports a limited amount of colors.)
4. If the output device supports RGB output, Convert the angle to a RGB.

4 Business Rules

- Implementations SHOULD allow the user to turn off any colorization completely.
- Implementations SHOULD implement the Color Vision Deficiency profiles and SHOULD allow the user to choose any of these profiles or to disable the correction.
- Implementations MUST NOT share the Color Vision Deficiency correction settings with other entities.

5 Algorithms

The algorithms in this document use the HSLuv¹ color space. It provides consistent brightness (for a given luminosity) across its entire definition space. There is also widespread library support.

5.1 Angle generation

Input: An identifier, encoded as octets of UTF-8 (RFC 3269²).

Output: Hue angle.

Note: The goal of this algorithm is to convert arbitrary text into a scalar value which can then be used to calculate a color.

1. Run the input through SHA-1 (RFC 3174³).
2. Treat the output as little endian and extract the least-significant 16 bits. (These are the first two bytes of the output, with the second byte being the most significant one.)
3. Divide the value by 65536 (use float division) and multiply it by 360 (to map it to degrees in a full circle).

¹HSLuv <<http://www.hsluv.org/>>.

²RFC 3269: UTF-8, a transformation format of ISO 10646 <<http://tools.ietf.org/html/rfc3269>>.

³RFC 3174: US Secure Hash Algorithm 1 (SHA1) <<http://tools.ietf.org/html/rfc3174>>.

5.2 Corrections for Color Vision Deficiencies

Input: Hue angle.

Output: Hue angle.

Note: This algorithm will re-map the angle to map it away from ranges which can not be distinguished by people with the respective Color Vision Deficiencies.

Note: Some floating-point modulo implementations will return negative outputs for negative inputs. This algorithm assumes that your implementation returns *non-negative* outputs for all inputs.

5.2.1 Red/Green-blindness

Add 90 to the angle, take it modulo 180 and subtract 90. Take the result modulo 360 to ensure that it's in the range from 0 to 360.

Note: the same effect can be achieved by forcing the two most-significant bits of the angle to be equal to the second-most-significant bit before converting to a float in [Angle generation](#). This avoids having to perform a floating-point modulo operation.

5.2.2 Blue-blindness

Take the angle modulo 180.

Note: the same effect can be achieved by setting the most-significant bit to zero before conversion to floating point in [Angle generation](#). This avoids having to perform a floating-point modulo operation.

5.3 RGB generation

Use the HSLuv operation `hsluvToRgb` to convert the Hue angle to a color. The saturation and lightness are to be defined by the implementation (see also the Contrast Ratio considerations).

5.4 Conversion of an RGB color palette to a Hue palette

Input: A set of RGB colors (each component from 0 to 1).

Output: A mapping from angles (integer, from 0 to 360) to RGB colors.

Note: when the algorithm finishes, the mapping maps angles (rounded to two decimal places) to the R, G, B triples which come closest to the desired color and lightness.

1. Create an empty mapping M which maps from Hue angles to quadruples of L, R, G and B.
2. For each color R, G, B from the input palette:

- a) If the R, G and B values are equal, skip the color and continue with the next. (Grayscale does not work well, since its saturation and hue are undefined.)
 - b) Calculate H, S and L from R, G, B using HSLuv.
 - c) Round the angle to the next integer value.
 - d) If the angle is not in the mapping M yet, or if the L value of the existing entry is farther away from 73.2 than the new L value, put the L, R, G, and B values as value for the angle into the mapping.
3. Strip the L values from the values of mapping M.
 4. Return M as the result of the algorithm.

Implementations are free to choose a representation for palette colors different from R, G, B triplets. The exact representation does not matter, as long as it can be converted to a Hue angle accordingly.

5.5 Mapping of a Hue angle to closest palette color

Input: (a) A mapping which maps angles to R, G, B triplets and (b) a color to map to the closest palette color as angle alpha.

Output: A palette color as R, G, B triplet.

Note: See [Conversion of an RGB color palette to a Hue palette](#) on how to convert an R, G, B triplet to an angle.

1. First, check if alpha rounded to an integer. If so, return that match immediately.
2. For each angle beta in the palette, calculate the distance metric:

$$D = \min((\alpha - \beta) \% 360, (\beta - \alpha) \% 360)$$

3. Return the R, G, B triplet associated with the angle with the smallest distance metric D.

Implementations are free to choose a representation for palette colors different from R, G, B triplets. The exact representation does not matter, as long as it can be converted to a Hue angle accordingly.

6 Implementation Notes

6.1 Gamma Correction

Implementations should be aware of Gamma correction and apply it as needed.

6.2 Normalization

When processing JIDs as text input, implementations **MUST** prepare the JID as it would for comparing it to another JID with a case-sensitive comparison function.

7 Accessibility Considerations

7.1 Color Vision Deficiencies

As outlined above, implementations **SHOULD** offer the *Red/Green-Blindness* and *Blue-Blindness* corrections as defined in the [Corrections for Color Vision Deficiencies](#) section. Users **SHOULD** be allowed to choose between:

- disabling all corrections (skip the Corrections for Color Vision Deficiencies step entirely),
- applying one of the Color Vision Deficiency correction profiles and
- disabling colorization altogether.

The last option is useful for users with monochromatic view or who find colors distracting. Some sources on the internet indicate that people with Color Vision Deficiencies may profit from having larger areas of color to be able to recognize them. This should be taken into consideration when selecting font weights and line widths for colored parts.

7.2 Contrast Ratio

Implementations should adapt the lightness value according to the background on which the color is rendered.

8 Security Considerations

This specification extracts a bit more information from an entity and shows it alongside the existing information to the user. As the algorithm is likely to produce different colors for look-alikes (see [Best Practices to Prevent JID Mimicking \(XEP-0165\)](#)⁴ for examples) in JIDs, it may add additional protection against attacks based on those.

Due to the limited set of distinguishable colors and only extracting 16 bits of the hash function output, possible Color Vision Deficiencies and/or use of palettes, entities **MUST NOT** rely on colors being unique in any context.

⁴XEP-0165: Best Practices to Prevent JID Mimicking <<https://xmpp.org/extensions/xep-0165.html>>.

9 Design Considerations

This section provides an overview of design considerations made while writing this specification. It shows alternatives which have been considered, and eventually rejected.

9.1 The YCbCr color space

The versions up to 0.5 of this document used a variant of the YCbCr color space (namely [BT.601⁵](#)) along with a custom algorithm to convert from angles to CbCr and from there to RGB. The HSLuv color space provides extremely consistent apparent brightness of the colors which cannot be achieved with simple application of YCbCr. In addition, HSLuv has widespread library support.

9.2 Hue-Saturation-Value/Lightness color space

The HSV and HSL color spaces fail to provide uniform luminosity with fixed value/lightness and saturation parameters. Adapting those parameters for uniform luminosity across the hue range would have complicated the algorithm with little to no gain.

9.3 Palette-based and context-aware coloring

Given a fixed-size and finite palette of colors, it would be possible to ensure that, until the number of entities to color exceeds the number of colors, no color collisions happen.

There are issues with this approach when the set of entities is dynamic. In such cases, it is possible that an entity changes its associated color (for example by re-joining a colored group chat), which defeats the original purpose.

In addition, more state needs to be taken into account, increasing the complexity of choosing a color.

9.4 Choice of mixing function in angle generation

This specification needs to collapse an arbitrarily long string into just a few bits (the angle in the CbCr plane). To do so, SHA-1 ([RFC 3174⁶](#)) is used.

CRC32 and Adler32 have been considered as faster alternatives. Downsides of these functions:

- Bad mixing without additional entropy.
- Adler32 is rarely available in standard libraries.

⁵BT.601: Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios <<https://www.itu.int/rec/R-REC-BT.601-7-201103-I/en>>

⁶RFC 3174: US Secure Hash Algorithm 1 (SHA1) <<http://tools.ietf.org/html/rfc3174>>.

- CRC32 is ambiguous: there are multiple polynomials in widespread use (e.g. the Ethernet and the zlib polynomials). Often it is not clear which polynomial is used by a library.

SHA-1 is widely available. From a security point of view, the exact choice of hash function does not matter here, since it is truncated to 16 bits. At this length, any cryptographic hash function is weak.

9.5 Palette-mapping function

The palette-mapping algorithm operates on angles only and disregards the lightness value except if the angles match. This has the downside that the brightness is not equal over the range of the palette mapped colors.

The alternative would be to require the lightness to be close to the target lightness. This has several issues:

- We cannot know if a palette can satisfy the given lightness at all.
- Many colors from e.g. the "Web Safe" palette (used in 256 color terminals and the test vectors) will not satisfy any given lightness, reducing the size of the effective palette drastically.

For the sake of having more colors available, the given algorithm was chosen which prefers many colors with hue conformance over fewer colors with hue and lightness conformance.

10 IANA Considerations

This document requires no interaction with the [Internet Assigned Numbers Authority \(IANA\)](#)⁷.

11 XMPP Registrar Considerations

This document requires no interaction with the [XMPP Registrar](#)⁸.

⁷The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <http://www.iana.org/>.

⁸The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.

12 Acknowledgements

Thanks to Klaus Herberth, Daniel Gultsch, Georg Lukas, Tobias Markmann, Christian Schudt, and Marcus Waldvogel for their input and feedback on this document.

13 Test Vectors

13.1 Test Vectors

This section holds test vectors for the different configurations. The test vectors are provided as Comma Separated Values. Strings are enclosed by single quotes ('). The first line contains a header. Each row contains, in that order, the original text, the text encoded as UTF-8 as hexadecimal octets, the angle in degrees, the calculated hue in degrees (differs from angle only for CVD-corrected rows), and the Red, Green, and Blue values.

13.1.1 No Color Vision Deficiency correction

```
text,hextext,angle,hue,r,g,b
'Romeo','526f6d656f',327.255249,327.255249,0.865,0.000,0.686
'juliet@capulet.lit','6a756c69657440636170756c65742e6c6974'
,209.410400,209.410400,0.000,0.515,0.573
' ','f09f98ba',331.199341,331.199341,0.872,0.000,0.659
'council','636f756e63696c',359.994507,359.994507,0.918,0.000,0.394
'Board','426f617264',171.430664,171.430664,0.000,0.527,0.457
```

13.1.2 With Red/Green-blindness correction

```
text,hextext,angle,hue,r,g,b
'Romeo','526f6d656f',327.255249,327.255249,0.865,0.000,0.686
'juliet@capulet.lit','6a756c69657440636170756c65742e6c6974'
,209.410400,29.410400,0.742,0.359,0.000
' ','f09f98ba',331.199341,331.199341,0.872,0.000,0.659
'council','636f756e63696c',359.994507,359.994507,0.918,0.000,0.394
'Board','426f617264',171.430664,351.430664,0.904,0.000,0.494
```

13.1.3 With Blue-blindness correction

```
text,hextext,angle,hue,r,g,b
'Romeo','526f6d656f',327.255249,147.255249,0.000,0.535,0.350
'juliet@capulet.lit','6a756c69657440636170756c65742e6c6974'
,209.410400,29.410400,0.742,0.359,0.000
' ','f09f98ba',331.199341,151.199341,0.000,0.533,0.373
```

```
'council', '636f756e63696c', 359.994507, 179.994507, 0.000, 0.524, 0.485
'Board', '426f617264', 171.430664, 171.430664, 0.000, 0.527, 0.457
```

13.2 Test Vectors for mapping to 216 color palette

The used palette can be generated by sampling the RGB cube evenly with six samples on each axis (resulting in 210 colors (grayscales are excluded)). The resulting palette is commonly known as the palette of so-called "Web Safe" colors.

13.2.1 No Color Vision Deficiency correction

```
text, hextext, hue, best_hue, r, g, b
'Romeo', '526f6d656f', 327.255249, 327, 1.000, 0.000, 0.800
'juliet@capulet.lit', '6a756c69657440636170756c65742e6c6974',
, 209.410400, 226, 0.000, 0.800, 1.000
' ', 'f09f98ba', 331.199341, 331, 1.000, 0.400, 0.800
'council', '636f756e63696c', 359.994507, 359, 0.800, 0.200, 0.400
'Board', '426f617264', 171.430664, 161, 0.000, 1.000, 0.800
```

13.2.2 With Red/Green-blindness correction

```
text, hextext, hue, best_hue, r, g, b
'Romeo', '526f6d656f', 327.255249, 327, 1.000, 0.000, 0.800
'juliet@capulet.lit', '6a756c69657440636170756c65742e6c6974',
, 29.410400, 30, 1.000, 0.600, 0.400
' ', 'f09f98ba', 331.199341, 331, 1.000, 0.400, 0.800
'council', '636f756e63696c', 359.994507, 359, 0.800, 0.200, 0.400
'Board', '426f617264', 351.430664, 353, 0.400, 0.000, 0.200
```

13.2.3 With Blue-blindness correction

```
text, hextext, hue, best_hue, r, g, b
'Romeo', '526f6d656f', 147.255249, 147, 0.400, 0.800, 0.600
'juliet@capulet.lit', '6a756c69657440636170756c65742e6c6974',
, 29.410400, 30, 1.000, 0.600, 0.400
' ', 'f09f98ba', 151.199341, 153, 0.200, 0.800, 0.600
'council', '636f756e63696c', 179.994507, 192, 0.000, 0.800, 0.800
'Board', '426f617264', 171.430664, 161, 0.000, 1.000, 0.800
```